TITLE

A METHOD OF TRANSFORMING VARIABLE LOOPS INTO CONSTANT LOOPS

CROSS REFERENCE TO RELATED APPLICATIONS

N/A

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

N/A

BACKGROUND OF THE INVENTION

The present invention relates generally to optimization of computer programs written in Hardware Description Languages (HDLs), and more specifically to a method of transforming variable loops into constant loops.

In modern circuit design, particularly in the development of ASICs (Application Specific Integrated Circuits) and FPGAs (Field Programmable Gate Arrays), extensive use is made of models written using HDLs. Examples of tools for processing HDLs in use today include DesignCompiler™ of Synopsis, Inc., and DesignVerifier™ of Chrysalis. Examples of HDLs in use today include Verilog and VHDL (Very high speed integrated circuits HDL). In a typical design process, a hardware designer writes an HDL program representing the high level operation and/or design of a circuit. This

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

HDL program may then be compiled and executed in order to test and verify the circuit design. Compilation of an HDL program is also sometimes referred to as "synthesizing" the HDL into a lower level representation of the circuit. The compiler for an HDL program is a type of computer aided design (CAD) system.

The compiled output of a CAD tool used to process an HDL program might describe a physical layout of gates which implement the circuit described by the HDL input. For example, in some systems, HDL is processed by what is referred to as a "gate synthesizer" program, the output of which is a "netlist" describing a network of gates within a physical circuit. The netlist output by a gate synthesizer may itself be used to test potential circuit designs, or even as the basis for automatic fabrication of actual circuits.

Circuit models written in HDL often involve looping statements. Illustrative types of looping statements are shown in Fig. 1 and Fig. 2. A "FOR" loop 10 looping statement is depicted in Fig. 1, and a "WHILE" loop 12 looping statement is shown in Fig. 2. The looping statements shown in Fig. 1 and Fig. 2 are specific examples of the many types of looping statements which may be employed to represent the structure of a circuit using an HDL.

In general, looping statements contain an initial expression, an exit expression, and an increment/decrement expression, as well as a loop body consisting of one or more of statements. For purposes of illustration, a generalized FOR loop statement 14 is

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

shown in Fig. 3. The FOR loop statement 14 of Fig. 3 is shown including initial expression INIT 16, exit expression EXIT 18, increment/decrement expression INC 20, and a loop body BODY_OF_STATEMENTS 22.

Looping statements may be categorized into what are referred to as constant looping statements and variable looping statements. A looping statement is a constant looping statement if both the initial expression and the exit expression involve only constants and the loop index. If the initial expression and exit expression of a looping statement do not both involve only constants and the loop index, then that looping statement is considered a variable looping statement. The looping statement 10 of Fig. 1 is an example of a constant looping statement, and the looping statement 12 of Fig. 2 is an example of a variable looping statement.

Hardware models developed using HDLs are often compiled prior to execution. Many HDL compilers perform a type of compile time optimization known as "loop unrolling". Loop unrolling is a process in which loop overhead can be reduced by expressly replicating the body of the loop within the program, thus reducing the iterations through the loop. In some cases, a looping statement may be completely eliminated by replicating the loop body a number of times dictated by the initial, exit, and increment/decrement expressions of the loop statement. Loop unrolling simplifies compilation and makes it possible for many programs to be executed more efficiently. In particular, loop unrolling is known to

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

enable greater degrees of program parallelization, thus potentially improving program execution speed.

Now considering Fig. 1, which is shown including a loop body 11, a loop unrolling optimization would transform the original looping statement 10 into the equivalent statements 24 of Fig. 4, completely eliminating the loop. As shown in Fig. 4, the loop index i from the original looping statement 10 of Fig. 1 is replaced by constants as the loop is unrolled. Based on the construction of the looping statement 10, the statements within the loop body 11 need to be performed twice.

A significant problem in existing computer aided design (CAD) systems for hardware circuit design, is that they fail to provide loop unrolling for variable looping statements. In particular, the above mentioned tools for processing HDLs - DesignCompiler™ of Synopsis, Inc., and DesignVerifier™ of Chrysalis, are unable to perform straight-forward loop unrolling for variable looping statements. Accordingly, such existing systems cannot perform the transformation from the looping statement 10 of Fig. 1 to the statements 24 in Fig. 4 for variable looping statements, such as the looping statement 12 of Fig. 2. This problem arises from the fact that the exact number of times the loop body of a variable looping statement must be replicated cannot be determined at compilation time. Thus, variable looping statements are often not synthesizable into netlists or other circuit models developed from HDLs.

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

For these reasons, it would be desirable to have a system for processing HDL programs which is capable of unrolling variable looping statements.

BRIEF SUMMARY OF THE INVENTION

Consistent with the principles of the present invention, a system and a method for processing variable looping statements in order to perform loop unrolling are disclosed. The disclosed system and method include a technique for transforming a variable looping statement into a constant looping statement. In order to form the constant looping statement, a range of loop index values is determined, including at least the complete set of values the loop index in the variable looping statement may take on within the variable looping statement. In an illustrative embodiment, the determined range of loop index values is derived from the types of the variables used in the initial and exit expressions of the variable looping statement. An upper bound and a lower bound are generated reflecting the determined range of loop index values.

The constant looping statement formed by the disclosed system and method includes a loop index that varies between the upper bound and lower bound derived from the variable looping statement. The generated constant looping statement further includes a nested control statement, such as an "if" statement, which predicates execution of the body of the constant looping

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

statement on a condition being satisfied. The condition checked within the nested control statement logically verifies that any conditions indicated by the initial expression and/or exit expression of the variable looping statement are met. Accordingly, while the generated upper and lower bounds may cause the loop index to assume values outside the range of values it would have taken on during execution of the original variable looping statement, the body of the generated constant looping statement is executed only in the event that conditions in both the initial expression and exit expression of the variable looping statement are satisfied. This advantageously allows use of upper and lower bounds in the generated constant looping statement that are not strictly "tight" without causing incorrect execution. In other words, the lower bound may be lower than the lowest possible loop index value in the variable looping statement, and the upper bound may be higher than the highest possible loop index value in the variable looping statement, without causing an incorrect loop index value to be applied to the body of the generated constant looping statement. The generated constant looping statement may then be unrolled in order to enable the various optimizations afforded by loop unrolling, such as increased parallelism during execution.

In an illustrative embodiment, the disclosed system is applicable to systems for processing HDL, such as HDL compilers. Since HDL models often are used to represent digital circuit designs, variables used within HDL programs typically represent lines or interfaces for

electronically conveying binary logic signals between hardware components. Such interfaces may be individual lines, or groups of lines as in a data bus. Accordingly, variables used within an HDL model are often defined as "bit field" types of variables, within which each "bit" may take on the value 0 or 1, representing the state of an individual line or signal. The range of values bit field variables may take on is relatively limited, in comparison with the ranges of potential values for other types of variables. Due to the prevalence of bit field variables within HDL models, the initial expression and exit expression of variable looping statements in such models often involve comparison of the loop index with bit field variables. As a result, the determination of an upper and lower bound for a loop index may be based on the absolute ranges of the variables involved, without the risk of highly inefficient results. Moreover, the nested conditional statement employed by the disclosed system within the generated constant looping statement limits the number of unnecessary statements performed as a result of using bounds that are beyond the actual range of loop index values applied to the loop body. For these reasons, the disclosed system effectively provides a reasonably efficient technique for providing the benefits of loop unrolling to variable looping statements contained within programs such as programs written in an HDL.

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

The invention will be more fully understood by reference to the following detailed description of the invention in conjunction with the drawings, of which:

Fig. 1 shows an illustrative constant looping statement;

Fig. 2 shows an illustrative variable looping statement;

Fig. 3 shows a generalized looping statement;

Fig. 4 illustrates an unrolled looping statement;

Fig. 5 is a flow chart illustrating steps performed in an illustrative embodiment;

Fig. 6 illustrates a generalized constant looping statement generated by an illustrative embodiment;

Fig. 7 shows another example of a variable looping statement;

Fig. 8 shows a constant looping statement generated by an illustrative embodiment in response to the variable looping statement of Fig. 7; and

Fig. 9 shows a constant looping statement generated by an illustrative embodiment in response to the variable looping statement of Fig. 2.

DETAILED DESCRIPTION OF THE INVENTION

As shown in Fig. 5, in an illustrative embodiment, a series of steps are performed in order to process a variable looping statement such that the variable looping statement is transformed into a constant looping statement on which loop unrolling may be performed. The

steps illustrated in Fig. 5 may be embodied within any specific type of computer program stored on a computer readable medium, and capable of execution on one or more processors within a computer system. For example, an executable program operable to perform the steps shown in Fig. 5 may be stored within a computer memory or program storage device that is communicably coupled to one or more processors operable to execute the computer program.

At step 52, the disclosed system determines an upper bound and a lower bound to be used to define a range of values for a loop index in a generated constant looping statement. The disclosed system determines the upper bound and lower bound at step 52 such that the range of values between the upper bound and lower bound includes all values that could be stored within the looping index of the variable looping statement. However, the upper bound and lower bound determined at step 52 need not include only those values that the looping index of the variable looping statement could take on. For example, the upper bound may be higher than the highest value that the looping index of the variable looping statement could receive. Similarly, the lower bound may be lower than the lowest value that the loop index of the variable looping statement could take on.

As part of the determination of the upper bound and lower bound at step 52, the disclosed system further determines a direction of the loop index within the variable looping statement. For example, the direction of the loop index in the variable looping statement is determined to be "up" in the event that the increment

expression of the variable looping statement causes the loop index to increase. Such a loop index is also referred to as an "increasing" loop index. Similarly, the direction of the loop index in the variable looping statement is determined to be "down" in the event that the increment expression in the variable looping statement causes the loop index to decrease. This type of loop index is referred to as a "decreasing" loop index.

In an illustrative embodiment, the disclosed system examines the type of the loop index and the other variable or variables within the initial and exit conditions of the variable looping statement to determine the upper bound and lower bound at step 52. In both the case of a variable looping statement having an increasing loop index and the case of a variable looping statement having a decreasing loop index, the disclosed system may use the type(s) of the variables compared to the loop index to determine the upper bound and lower bound in step 52. However, in either the case of an increasing or decreasing loop index, if the type of the loop index allows storage of a more restricted range of values than the type(s) of one or more of the variables being compared with the loop index, then the value of one or both of the upper and lower bound may be determined based on the type of the loop index.

For example, in the case of a variable looping statement having an increasing loop index, the disclosed system may determine an upper bound at step 52 as being the highest value that can be stored in a variable that

is compared to the loop index within the exit expression of the variable looping statement, based on the type of that variable. However, if the highest value that may be stored in the loop index itself is less than the highest value that may be stored in the variable being compared to the loop index within the exit expression of the variable looping statement, the upper bound determined at step 52 may be the highest value that may be stored within the loop index itself.

Also in the case of a variable looping statement having an increasing loop index, the disclosed system may determine a lower bound at step 52 that is the lowest value that may be stored in a variable being compared to the loop index within the initial expression of the variable looping statement, based on the type of that variable. However, if the lowest value that may be stored in the loop index itself is greater than the lowest value that may be stored in the variable being compared to the loop index within the initial expression of the variable looping statement, the lower bound determined at step 52 may be the lowest value that may be stored within the loop index itself.

In the case where the disclosed system determines that the variable looping statement employs a decreasing loop index, then the disclosed system may determine an upper bound at step 52 that is the highest value that may be stored in a variable being compared to the loop index within the initial expression of the variable looping statement, based on the type of that variable. However, if the highest value that may be stored in the loop index

itself is less than the highest value that may be stored in the variable being compared to the loop index within the initial expression of the variable looping statement, then the upper bound determined at step 52 may be the highest value that may be stored within the loop index itself.

Also in the case of a variable looping statement having a decreasing loop index, the disclosed system may determine a lower bound at step 52 that is the lowest value that may be stored in a variable being compared to the loop index within the exit expression of the variable looping statement, based on the type of that variable. Again, in the case where the lowest value that may be stored in the loop index is greater than the lowest value that may be stored in the variable being compared to the loop index within the exit expression of the variable looping statement, the lower bound determined at step 52 may be the lowest value that may be stored within the loop index itself.

At step 54, the disclosed system determines a condition that is to be tested within the generated constant looping statement, such that the body of the generated looping statement is only executed when the condition is satisfied. In an illustrative embodiment, the condition determined at step 54 is tested within a nested control statement, such as an "if" statement, which predicates execution of the body of the generated constant looping statement on the condition being satisfied. The condition generated at step 54 logically verifies that any conditions indicated by the initial

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

expression and/or exit expression of the variable looping statement are met before the loop body of the generated constant looping statement is executed. For example, in the case of a variable looping statement in which the initial expression and exit expression of the variable looping statement both consist of expressions in which the loop index is compared with one or more variables, then the condition generated at step 54 would be the logical "AND" of both the initial expression and the exit expression of the variable looping statement. In the case where only one of the initial expression or exit expression of the variable looping statement involves comparison of the loop index with a variable, then the condition generated at step 54 need only include the one of the initial expression or exit expression including comparison of the loop index with the variable.

At step 56, the disclosed system forms a constant looping statement that is functionally equivalent to the input variable looping statement. The constant looping statement formed at step 56 includes a loop index that varies across a range of values defined by the lower and upper bounds determined at step 52, and also includes a control statement which tests the condition generated at step 54. The body of the constant looping statement formed at step 56 is thus only executed in the event that the condition generated at step 54 is satisfied. The body of the constant looping statement formed at step 56 includes the body of the variable looping statement. In order to ensure correct operation, the disclosed system thus employs the condition generated at step 54 to ensure

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

that the body of the generated constant looping statement is executed only in the event that conditions in both the initial expression and exit expression of the variable looping statement are satisfied. The constant looping statement formed at step 56 may then be unrolled in order to enable the various optimizations afforded in connection with loop unrolling. The below table summarizes key aspects of bounds generation for the constant looping statement in the illustrative embodiment:

CASE	BOUNDS DETERMINATION
Increasing Loop Index, and maximum possible value for comparison variable in exit expression is less than maximum possible value of Loop Index	Upper Bound is maximum possible value for comparison variable
Increasing Loop Index, and maximum possible value for comparison variable in exit expression is greater than or equal to maximum possible value of Loop Index	Upper Bound is maximum possible value for Loop Index
Decreasing Loop Index, and minimum possible value for comparison variable in exit expression is greater than minimum possible value of Loop Index	Lower Bound is minimum possible value for comparison value
Decreasing Loop Index, and minimum possible value for comparison variable in exit expression is less than or equal to minimum possible value of Loop Index	Lower Bound is minimum possible value for Loop Index

Figs. 6-9 further illustrate the disclosed transformation of a variable looping statement into an equivalent looping statement for purposes of loop unrolling optimization. Fig. 6 shows a generalized constant looping statement 60 as would be generated by an illustrative embodiment of the disclosed system in response to the generalized variable looping statement shown in Fig. 3 in the case where the loop index of the variable looping statement is an increasing loop index. The generalized constant looping statement 60 of Fig. 6 is shown including a LOWER_BOUND_EXPRESSION 62, an UPPER_BOUND_EXPRESSION 64, and an INCREMENT_EXPRESSION 66. The LOWER_BOUND_EXPRESSION 62 compares the loop index of the constant looping statement 60 with a lower bound determined in response to the INIT expression 16 shown in Fig. 3, as described above with reference to step 52 of Fig. 5. The UPPER_BOUND_EXPRESSION 64 compares the loop index of the constant looping statement 60 with an upper bound determined in response to the EXIT expression 18 shown in Fig. 3, as also described above with reference to step 52 of Fig. 5. The INCREMENT_EXPRESSION 66 increases the value of the loop index in the constant looping statement 60 just as the INC expression 20 increased the value of the loop index in the variable looping statement 14 in Fig. 3. The condition 68 consists of the logical AND of the INIT expression 16 and the EXIT expression 18 of the variable looping statement 60 of Fig. 3. Accordingly, the STATEMENT_BODY 69 is only executed in the event that both

the INIT expression 16 and the EXIT expression 18 are satisfied. The STATEMENT_BODY 69 includes the BODY_OF_STATEMENTS 22 of the variable looping statement 14 shown in Fig. 3.

Fig. 7 shows a variable looping statement 70. The initial expression 72 determines whether the loop index i is less than or equal to the value of the variable j. The exit expression 74 determines whether the loop index i is less than the value of the variable k. The increment expression 76 increments the loop index i following each execution of the loop body 78. In the example looping statement 70, the loop body is executed repeatedly while the initial expression 72 and exit expression 74 are true. The syntax of the for loop 70 shown is Fig. 7 is for purposes of illustration, and those skilled in the art will recognize that other specific for loop syntaxes exist that may also be processed by the disclosed system and method. Further in Fig. 7, the variable i is shown declared as a register type variable having four bits in the statement 80, variable j is shown declared as a register type variable having 2 bits in the statement 82, and the variable k is shown declared as a register type variable having 3 bits in the statement 84. In the example of Fig. 7, the register type variables i, j and k are illustrative of bit field variables. Those skilled in the art will recognize that other specific types of bit field variables are employed in various specific HDLs, and that the disclosed system is generally applicable to processing of variables of any type within initial and

exit expressions, such as other specific bit field variable types. In the example of Fig. 7, the disclosed system determines that the maximum value of the variable i in the for loop 70 is controlled by the type of the variable k, since the maximum value that can be stored in k is 7, which is less than the maximum value of 15 that may be stored in i.

Fig. 8 shows a constant looping statement 90 derived by the disclosed system and method from the variable looping statement 70 of Fig. 7. As shown in Fig. 8, the initial statement 92 assigns the value of i to a variable m, since the value of i entering the statement body not determined within the looping statement. The disclosed system further operates to replace all instances of i within the statement body with the variable m, to ensure correctness. The variable m thus becomes the loop index for the constant looping statement 90 of Fig. 8.

In the exit expression 94 shown in Fig. 8, the loop index m is compared with the constant value 7, since 7 is the highest value that the register variable k can store. The maximum value that the register variable k can store is a value in which each of the 3 bits in k are set, resulting in a value of 7. The increment expression 96 increments the loop index m after each iteration through the statement body 100, corresponding to the addition of 1 to the loop index in the increment expression 76 of Fig. 7. As also shown in Fig. 8, the constant looping statement 90 further includes a condition 98 that is tested before the statement body 100 is executed. If the condition 90 is not satisfied, then the statement

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

body 100 is not executed. The condition 98 is shown as the logical AND of the initial expression 72 and the exit expression 74 in the variable looping statement 70 of Fig. 7, except with the loop index i replaced by the loop index m.

Fig. 9 shows a constant looping statement 110 as would be generated by an illustrative embodiment of the disclosed system in response to the variable looping statement 12 shown in Fig. 2. In the constant looping statement 110, shown for purposes of illustration as a "while" loop, a test condition 112 determines whether the variable x is less than or equal to the constant 15. Thus it is shown that the disclosed system determined that the variable y in the variable looping statement 12 of Fig. 2 could not represent a value larger than 15, since y is a register variable that is 4 bits wide. The constant looping statement 110 further includes a condition 114 that is tested before the loop body 116 is executed. The condition 114 is the same as the test condition 13 in the variable looping statement 12 of Fig. 2.

Those skilled in the art should readily appreciate that the programs defining the functions of the disclosed system and method for processing variable looping statements can be implemented in and delivered to a specific embodiment of the disclosed system in many forms; including, but not limited to: (a) information permanently stored on non-writable storage media (e.g. read only memory devices within a computer such as ROM or CD-ROM disks readable by a computer I/O attachment); (b)

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313

information alterably stored on writable storage media (e.g. floppy disks and hard drives); or (c) information conveyed to a computer through communication media for example using baseband signaling or broadband signaling techniques, including carrier wave signaling techniques, such as over computer or telephone networks via a modem. In addition, while the illustrative embodiments are described as implemented in computer software, the functions within the illustrative embodiments may alternatively be embodied in part or in whole using hardware components such as Application Specific Integrated Circuits or other hardware, or some combination of hardware components and software.

While the invention is described through the above exemplary embodiments, it will be understood by those of ordinary skill in the art that modification to and variation of the illustrated embodiments may be made without departing from the inventive concepts herein disclosed. In particular, while some of the illustrative embodiments are described in connection with processing of FOR loops, the disclosed system and method are also applicable to any other specific types of control statements that execute a section of code a specified number of times. Accordingly, the actual syntax and usage of the variable looping statement processed by the disclosed system and method may vary from language to language. Similarly, while the illustrative embodiments are described in connection with use of a nested control statement consisting of an "if" conditional statement, the disclosed system and method may alternatively employ

any other specific types of conditional statements that affect the flow of execution through a program, such as a CASE statement. Accordingly, the invention should not be viewed as limited except by the scope and spirit of the appended claims.

ATTORNEY DOCKET NO. P5426
WEINGARTEN, SCHURGIN,
GAGNEBIN & HAYES LLP
TEL. (617) 542-2290
FAX. (617) 451-0313